A Catalogue of Green Architectural Tactics for the Cloud

Procaccianti, G.; Lago, P.; Lewis, G.A.

published in
International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA)
2014

document version
Peer reviewed version

A Catalogue of Green Architectural Tactics for the Cloud

Procaccianti, G., Lago, P., Lewis, G.A.

In: (Ed.), International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA). Victoria, CA: IEEE

A Catalogue of Green Architectural Tactics for the Cloud

Giuseppe Procaccianti*†
*VU University Amsterdam, the Netherlands
†Politecnico di Torino, Italy
g.procaccianti@vu.nl

Patricia Lago
VU University Amsterdam, the Netherlands
p.lago@vu.nl

Grace A. Lewis*‡
*VU University Amsterdam, The Netherlands
‡CMU Software Engineering Institute, USA
glewis@sei.cmu.edu

***Abstract*—Energy efficiency is a primary concern for the ICT sector. In particular, the widespread adoption of cloud computing technologies has drawn attention to the massive energy consumption of data centers. Although hardware constantly improves with respect to energy efficiency, this should also be a main concern for software. In previous work we analyzed the literature and elicited a set of techniques for addressing energy efficiency in cloud-based software architectures. In this work we codified these techniques in the form of Green Architectural Tactics. These tactics will help architects extend their design reasoning towards energy efficiency and to apply reusable solutions for greener software.**

I. Introduction

The global energy consumption of data centers and Internet infrastructure is predicted to consume approximately 18% of expected world power capacity by 2030 [1]. As the adoption of cloud computing technologies continues to grow, the need for energy-efficient solutions becomes evident. Nonetheless, cloud-based software holds great potential for energy efficiency (EE). A recent study [2] showed that migrating all business applications in the U.S. to the cloud could reduce their energy footprint by 87%. A previous work [3] started to analyze the cost and energy benefits of data migration to the cloud.

However, this transition to the cloud is not an easy task. Cloud-based software must be appropriately designed to address EE, which is typically not the case for traditional business applications. If these applications are abruptly migrated, it is highly likely that the resulting energy waste would significantly outweigh the expected benefits. For this reason, it is time for software engineers to take into account the energy implications of their design decisions.

Recent efforts of the software engineering community [4] have stressed the responsibility of software in the search for sustainability. One of the challenges that emerged for Green Software Engineering is finding generally reusable solutions for energy-efficient software design. In previous work we performed a Systematic Literature Review (SLR) [5] [6] to discover software architectural solutions for cloud-based software that addressed EE-related issues. The SLR identified a number of recurring techniques that were potentially reusable in other solutions. In this work we codify these techniques in the form of Green Architectural Tactics. These tactics can be adopted by software architects and developers during the design and development of cloud-based software systems or the refactoring of existing business applications for cloud migration. This contribution will support decision-making when dealing with EE aspects of cloud-based software architectures.

This work extends a paper published in [7], where we presented three scenarios for energy efficiency and an example tactic for each scenario. In this paper we extend our initial work, by presenting our view of energy efficiency as a quality attribute and a full catalogue of Green Architectural Tactics with examples.

The paper is structured as follows: in Section II we present similar approaches and efforts that address EE as an architectural concern. Section III introduces EE as a quality attribute. In Section IV the Green Architectural Tactics are presented and described with application examples extracted from the literature. Section V discusses the architectural implications of EE. Finally, Section VI presents our strategy for evaluating the impact of the Tactics and Section VII draws conclusions and states our future research goals.

II. Related Work

While a steadily growing scientific body is being built on green software engineering [8], most research focuses on estimating or measuring power consumption at the system- or source-code level, without suggesting ways to actually develop energy-aware software (e.g. [2], [3]). Very little research has been carried out in studying EE at the software architecture level, neither in general nor for cloud-based software. Some preliminary investigations go back to the work of Rangaraj & Bahsoon [9], who used market-based economics theory to define a framework for optimizing power consumption in energy-unaware software architectures at runtime. Bahsoon then planned to apply the same approach to cloud architectures [10]. In [11], Seo et al. come closer to our objective by defining a framework that estimates the energy consumption of three distributed system architectural styles. Their goal is to evaluate the most appropriate architectural style *before* implementation. Te Brinke et al. [12] propose a design method to extend modules with energy optimizers. Although Rangaraj

and Bahsoon agree with us in considering architecture the right abstraction level for addressing energy-related concerns [9], no work so far has provided support for architects to actually design software architectures that address EE upfront.

For software system architectures the story is not that different. For example, EE in mobile computing is a widely addressed topic because of battery limitations of mobile devices [13]. Cyber-foraging, a form of mobile cloud computing in which mobile devices offload expensive computation to more powerful servers in the cloud, is a common strategy for saving battery power on mobile devices [14]. However, it is not uncommon for literature on cyber-foraging to refer to the cloud as having infinite resources; which means that no reusable cyber-foraging strategies have been defined yet for architecting energy-aware software systems that address the EE of the system as a whole.

That being said, there are some existing tools that can be used to implement tactics for EE in Cloud-based software. Amazon Web Services (AWS), for example, provides an Auto Scaling[1] feature that scales the capacity of VM instances (EC2, Elastic Compute Cloud) elastically depending on user-defined conditions. In addition, Amazon also provides CloudWatch[2], a Web Service that monitors several metrics of the EC2 instances that can be used to trigger Scaling operations. These tools can be used to implement either Energy Monitoring or Self-Adaptation tactics for EE, later described in this contribution.

III. Energy Efficiency as a Quality Attribute

According to Bass et al. [15], EE is to be regarded as a "system" quality attribute because it is the result of an indirect action of software. However, Bass et al. also argue that the line between "software" and "system" quality attributes is very thin. In the end, even if energy is ultimately consumed by hardware, it is software that determines hardware behavior. In order to provide a clear representation of EE as a quality attribute, we follow the approach adopted by Bass et al. [15] and characterize EE through *quality attribute scenarios*. We grouped our Green Architectural Tactics in three categories and formulated a scenario for each category (see Table I). In all the scenarios, the *response measure* is energy consumption values. In the following section, we describe the identified scenarios for each category, as well as the elicited Green Architectural Tactics.

IV. Green Architectural Tactics

In previous work [5] we identified a set of recurring design solutions, described in the literature, to achieve EE in cloud-based software architectures. In this work, we codified these solutions as *tactics* – that is, "design decisions that influence the achievement of a quality attribute response" [15]. Each tactic is described in terms of:

- *Motivation:* rationale behind the tactic.
- *Description:* components introduced by the tactic and their roles.
- *Constraints:* necessary conditions for applying the tactic in an existing software architecture.
- *Example:* previous application of the tactic.
- *Dependencies:* whether the tactic requires other tactics to be applied.

In the following, we describe the identified scenarios for each category, as well as an example of a Green Architectural Tactic.

A. Energy Monitoring

A typical scenario for EE that involves Energy Monitoring is the following: the system administrator of a cloud-based system wants to know the energy consumption of its infrastructure during operations. The Energy Monitor gathers the energy consumption information and presents it to the administrator.

The tactics in this category are targeted at monitoring the energy consumption of the cloud infrastructure. These tactics are often combined with tactics from other categories; Self-Adaptation in particular because information from monitoring components is typically used to trigger adaptive mechanisms.

*1) **Metering**:*

Motivation. Instrumenting a data center with power metering devices is becoming common practice[3]. The market is flooded with many different models of power meters with enhanced capabilities (e.g., wireless communications, high sampling frequencies, data analysis features). Many devices come with built-in sensors and tools to monitor power consumption in real-time. The Metering tactic enables to effectively use the information provided by these devices.

Description. The Metering tactic consists of collecting power metering information from the hardware through dedicated software components called Energy Collectors. Collectors are usually in a many-to-many relationship with physical power meters. These Collectors share information via an Energy Communication Bus (ECB) that provides a common interface for energy information. In addition, the energy consumption information is stored in a dedicated Energy Database that can have different levels of granularity. Finally, a GUI component called an Energy Dashboard provides graphical representations of energy information along with useful reporting for both cloud service providers and customers.

Constraints. The main limitation of this tactic is the need for a physical metering infrastructure, which can be costly in the case of large data centers. In addition, the granularity of the information gathered and shared by the metering process has to be tuned accordingly in order to avoid information overload.

Example. An example of how to apply the Metering tactic is shown in the CompatibleOne project [16]. CompatibleOne is a cloud resource management software that allows the creation of hybrid cloud platforms through the aggregation of services from different cloud providers. In this context, an Energy Monitoring framework was developed to monitor the energy consumption of each cloud provider participating in the platform for subsequent energy billing and environmental impact evaluation. Several power meters and probes are supported by the framework. As shown in Figure 1a, starting from the hardware layer at the bottom of the figure, the power consumption data flows from the physical resources through the probes. A Collecting Daemon, of type Energy Collector, retrieves the data through the GetValue interface of

[1] http://aws.amazon.com/autoscaling/

[2] http://aws.amazon.com/cloudwatch/

[3] http://www.greenbiz.com/news/2009/04/14/whys-and-hows-measuring-power-your-data-center, last visited on October 1st, 2013.

TABLE I. QUALITY ATTRIBUTE SCENARIOS FOR ENERGY EFFICIENCY AND TACTICS OVERVIEW

Energy Efficiency Scenarios			
Category	*Energy Monitoring*	*Self-Adaptation*	*Cloud Federation*
Stimulus	Request for energy consumption information	Energy consumption alert	Energy consumption alert
Source of Stimulus	Administrator	Energy Monitor	Energy Monitor
Environment	Normal operation	Runtime	Multi-cloud
Artifact	Energy Monitor	Hypervisor	Orchestrator
Response	The Energy Monitor presents the detailed energy consumption information for the data center.	The Hypervisor consolidates the VMs on the less-active servers and then shuts down the idle servers.	The Orchestrator swaps the most energy-consuming services with less energy-consuming services.
Response Measure	Energy consumption values	Energy consumption values	Energy consumption values
Tactics	Metering Static Classification Modeling	Scaling Down Consolidation Workload Scheduling	Energy Brokering Service-Adaptation

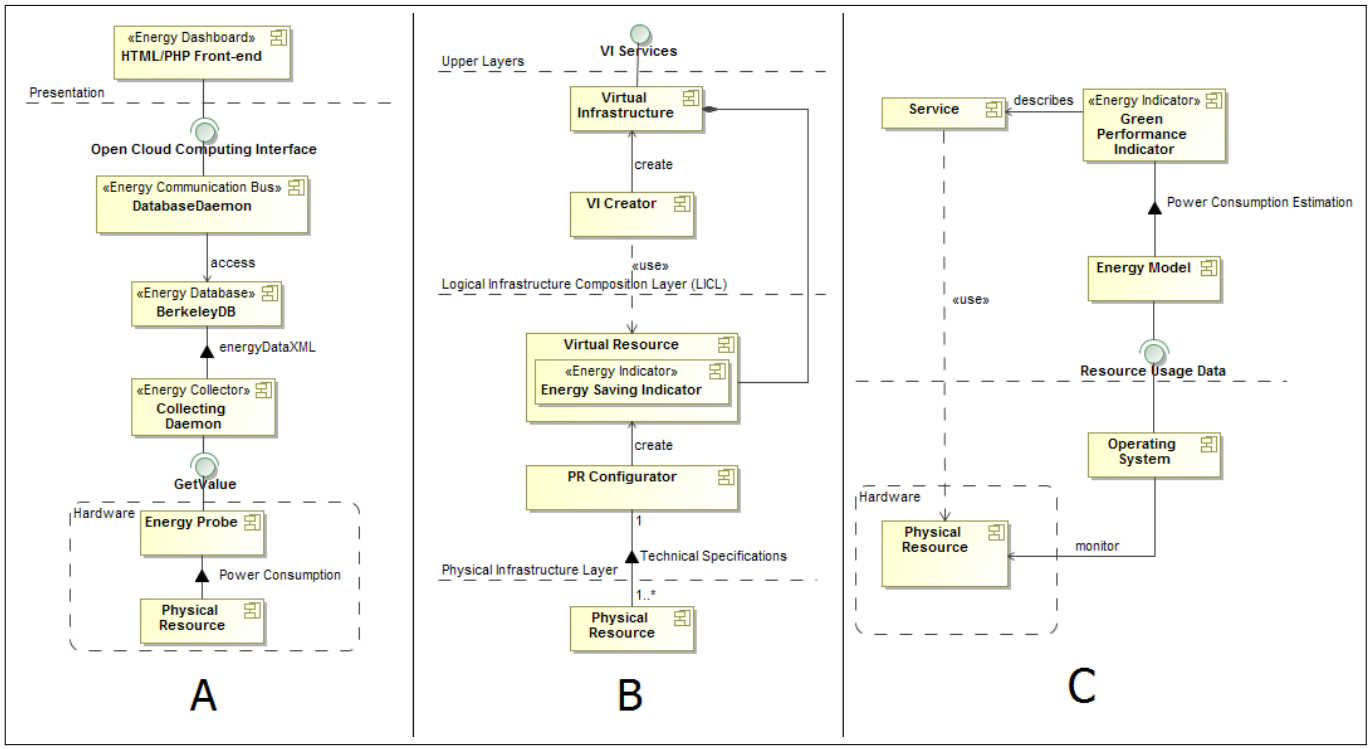

Fig. 1. A. Example of the Metering tactic. B. Example of the Static Classification tactic. C. Example of the Modeling tactic.

the probes. This data is then converted into XML format and stored in a BerkeleyDB database (Energy Database). Another software component, a DatabaseDaemon that acts as an Energy Communication Bus, provides access to the database to HTML and PHP front-ends (Energy Dashboard) through an Open Cloud Computing Interface, a standard set of specifications for cloud computing providers. Finally, the front-ends present the information to the system administrators.

*2) **Static Classification**:*

Motivation. A cloud infrastructure is typically composed of many heterogeneous IT devices. Direct energy consumption monitoring of each one of these devices might be infeasible because the physical machines might be external to the organization of the cloud software provider. The Static Classification tactic provides a solution to estimate the power consumption of the infrastructure when metering information is unavailable.

Description. This tactic consists of classifying the different resources in terms of EE through the use of Energy Indicators. This classification is static, i.e., not based on on-line, real-time information, but rather on technical specifications and characteristics of the devices themselves. To some extent, the Energy Indicators share an analogy with the Energy Labels[4] designed by the EU to classify the EE of appliances.

Constraints. Unfortunately, hardware vendors not always disclose energy consumption specifications of their products. In addition, this tactic is not applicable to any operation that requires an on-line analysis of software behavior on a fixed physical platform.

Example. An example of this tactic can be seen in the GEYSERS EU project [17]. The context is a multi-layered software architecture for dynamic cloud service provisioning in which the Physical Infrastructure Layer (PIL) is decoupled from the Logical Infrastructure Composition Layer (LICL). One of the goals of the project is the selection of the "greenest" physical resources, based on Static Classification, to create energy-efficient Virtual Infrastructures (VIs). The example is described in the component diagram in Figure 1b: the PR Configurator takes as input the technical specifications of physical resources

[4]http://www.newenergylabel.com/index.php, last accessed on September 18, 2013.

and assigns "Energy Saving Indicators" (ESIs), of type Energy Indicator, to created Virtual Resources. Subsequently, the VI Creator component in the LICL composes Virtual Resources into VIs, using the ESIs to prioritize the selection of the resources. Finally, the LICL exposes the services provided by the VIs to the upper layers of the architecture.

3) ***Modeling****:*

Motivation. In order to implement self-adaptive mechanisms it is necessary to have near-real-time energy consumption information. This enables the modification of software behavior according to how much energy the system is actually consuming. When metering systems are unavailable, the Modeling tactic is a viable option.

Description. The Modeling tactic enables a dynamic estimation of power consumption values through predictive Energy Models. These Models are embedded in Energy Indicators, similar to those in the Static Classification tactic. However, these Energy Indicators do not statically classify physical resources, but rather provide a dynamic estimation of the power consumption of the software components. Typically, Energy Models are built through regression analysis based on software runtime metrics, i.e. resource usage (CPU, disk, memory) [18].

Constraints. The limitation of this tactic lies in the accuracy of the software Energy Models. To date, many models and tools are available to estimate software energy consumption but their accuracy varies greatly based on the selected hardware platform. In addition, not all hardware resources are good predictors of energy consumption; identifying the best predictors is still an issue for researchers in the field [19].

Example. A prototype showing the application of this tactic is provided by de Oliveira et al. [20]. The context is a Service-Oriented Architecture (SOA) applied to a cloud infrastructure. As shown in Figure 1c, for each service of the SOA, the Operating System of each physical node provides service-related Resource Usage Data (in the case of the example, CPU, memory and disk [20]). A linear Energy Model retrieves this data and estimates the power consumption impact of each service. The estimation is modeled into a Green Performance Indicator (GPI), of type Energy Indicator. Each GPI describes a service in terms of EE.

B. Self-Adaptation

The Self-Adaptation scenario for EE starts from the Energy Monitor that reports an alert of excessive energy consumption while the system is not fully loaded. In response, the Cloud Hypervisor (i.e. the Virtual Machine Monitor [21]) migrates some of the VMs to less-loaded servers so that it can shut down the resulting idle servers.

Tactics in this category implement mechanisms that modify runtime software configurations for the specific purpose of lowering energy consumption. In cloud-based environments Self-Adaptation mostly concerns the configuration, deployment, and workload of Virtual Machines (VMs).

1) ***Scaling Down****:*

Motivation. One of the key features of cloud computing is the ability to provide resources on demand. When more resources are needed to satisfy incoming requests, the cloud infrastructure allocates more physical resources to VMs (*scaling up* or *vertical scaling*). However, the opposite mechanism should also be in place: when a decrease in demand occurs VMs must be appropriately scaled *down* in order to avoid energy waste. The Scaling Down tactic describes how to design this mechanism.

Description. An important component of this tactic is the Scale Unit, i.e., a pre-defined "block of IT resources" [22] explicitly modeled as a software component. Modeling Scale Units is useful for planning the scaling operations because it defines a finite number of configurations for the VMs. Thus, it is possible to associate each configuration with a particular level of demand or system load. The Adaptation Engine is the component that performs the Scaling operation; this role is typically played by the Hypervisor. Another key component is the SLA Violation Checker. During the Scaling operation the fulfillment of established service-level objectives must be ensured at all times. This component performs the needed checks and accordingly allows or disallows the Adaptation Engine to perform the Scaling.

Constraints. Scaling is a complex operation that requires careful planning and continuous monitoring. The main challenge is determining the right amount of resources that define a Scale Unit. This implies the prediction of expected levels of demand, which is not an easy task especially in large-scale cloud service provisioning.

Example. A possible implementation of the Scaling Down tactic is provided by Xu et al. [23]. As illustrated in Figure 2a, each Virtual Resource is configured by the Adaptation Engine, realized by the Effector and the Scheme Planner. The Effector actively executes the scaling of the Virtual Resources, by a number of Scale Units determined by the Scheme Planner that evaluates the current VM configuration considering the requirements of the system. In this example, Scale Units are modeled in terms of assigned virtual processors (vCPUs) and memory size (memoryGB). The Virtual Resources are used by the Cloud Application that exposes its service-level metrics via a REST API (SL Metrics). The CApp SLA Manager, of type SLA Violation Checker, monitors those metrics to ensure that service-level objectives are met. If necessary, the CApp SLA Manager issues a notification to the Scheme Planner to scale up the Virtual Resources again.

Dependencies. The Scaling Down tactic requires some sort of Energy Monitoring tactic for the Adaptation Engine to decide whether or not to perform scaling operations. In the previous example a Metering tactic is implemented by Sensors (Energy Collectors) that collect energy-related metrics; a Monitoring Center (Energy Monitor) that records, filters and audits the data provided by the sensors; and a Knowledge Base (Energy Database) where energy consumption information is stored.

2) ***Consolidation****:*

Motivation. As mentioned earlier, on-demand resource provisioning is an important feature of cloud-based environments. Adding resources to a single VM may not always be the best option. For example in cloud application server provisioning[5] creating new VM instances may provide additional flexibility and help to perform load balancing among servers. This is called *horizontal* scaling (or scaling *out*). This operation, however, may easily lead to inefficient usage of physical

[5]http://www.enterprisenetworkingplanet.com/netos/article.php/3753836/Practical-VM-Architecture-How-Do-You-Scale.htm, last visited on Oct. 2013

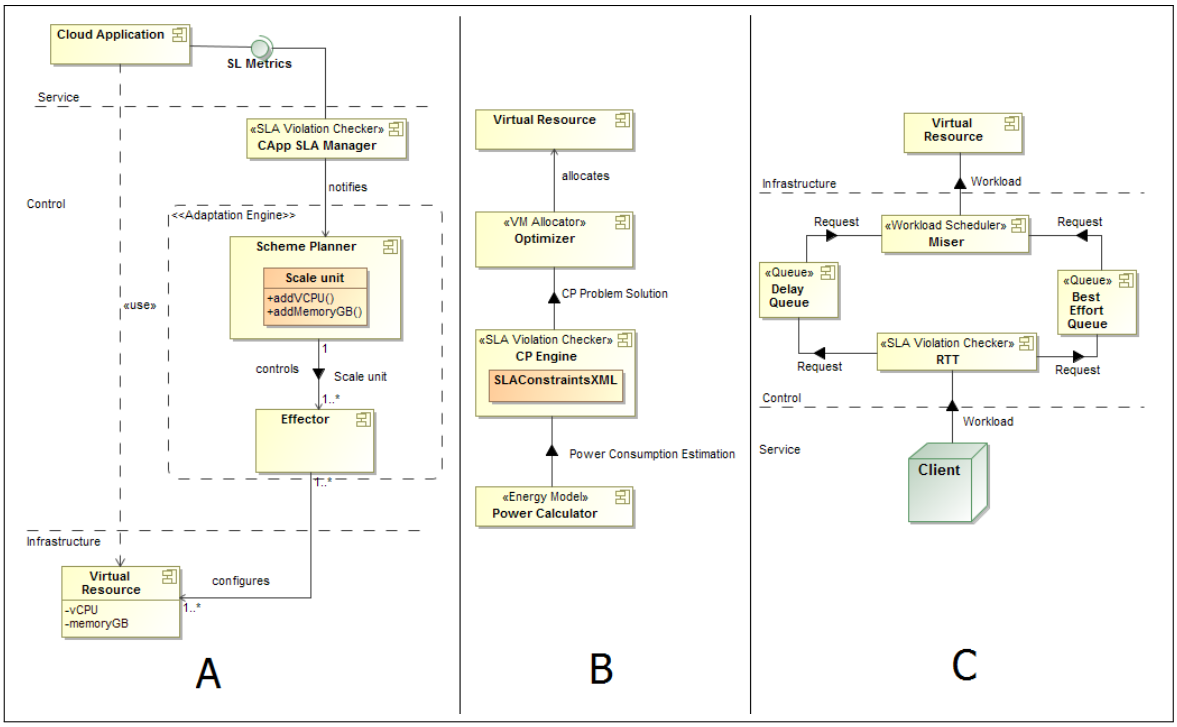

Fig. 2. A. Example of the Scaling Down tactic. B. Example of the Consolidation tactic. C. Example of the Workload Scheduling tactic.

resources if the density of VMs across the physical servers is not accurately managed in low-request phases. Indeed, the Consolidation tactic concentrates the VM instances on the minimum number of servers needed. Powering down the unused servers will evidently increase the EE of the cloud-based software.

Description. The main component of the Consolidation tactic is the VM Allocator, the software component responsible for live VM migration. This component can be (a part of) the Hypervisor, as in the Adaptation Engine in the Scaling tactic. The SLA Violation Checker is needed as well to check the fulfillment of service-level objectives after VM migrations.

Constraints. Consolidation must take place at runtime. This means that VMs must be represented in a format that allows them to be seamlessly migrated from one location to another, along with their context, workload, and metadata. This may introduce high network traffic and security risks.

Example. Dupont et al. [24] provide a sample implementation of the Consolidation tactic, depicted in Figure 2b. The Power Calculator, of type Energy Model, provides a power consumption estimation to the CP Engine, of type SLA Violation Checker. The CP Engine formulates a constraint programming problem using the constraints extracted from the SLAs in XML format (SLAConstraintsXML). The CP engine then solves the problem and the Optimizer (of type VM Allocator) produces a VM allocation scheme by applying the solution to the Virtual Resources.

Dependencies. The presence of the Power Calculator indicates a dependency on the Modeling tactic: as shown in Figure 2b, the Power Calculator is an instance of an Energy Model.

*3) **Workload Scheduling**:*

Motivation. The property of adapting to workload changes by provisioning and de-provisioning resources is called *elasticity* [25] and it is commonly regarded as a defining property of cloud environments. Elasticity has, of course, a direct connection with EE: the more closely resource provisioning matches demand, the more energy efficient the infrastructure is. The Scaling Down tactic allows to adapt resource provisioning, while the Workload Scheduling tactic is meant to prioritize and assign the load to the different virtual resources in order to match the demand.

Description. In this tactic, a Workload Scheduler is a software component that is able to dispatch workloads to VMs. The Scheduler normally uses one or more Queues to arrange the workloads. Queues can be differentiated in terms of priority levels, QoS requirements or deadlines. The SLA Violation Checker ensures that all service-level objectives are met.

Constraints. Workload scheduling is a well-known practice in software systems that is widely studied in operating systems theory. However, workload scheduling has specific challenges in cloud-based environments. First, when modeling workloads it is necessary to select the appropriate workload granularity. For example, a workload can divided per application, VM, or pool of VMs. In addition, efficient workload prediction in cloud environments is difficult to achieve because of the high variability of demand.

Example. Lu et al. [26] provide an example of Workload Scheduling for cloud storage services. In their solution, shown in Figure 2c, when a Client node submits a Workload to the service, the RTT algorithm, of type SLA Violation Checker, decomposes the Workload into Requests, to be assigned to different Queues, according the deadline of each Request. In the example, two Queues are present: a Delay Queue that has a guaranteed response time and a Best Effort Queue that has no time constraints. The Miser algorithm, of type Workload Scheduler, is used to recombine Workloads and dispatch them to Virtual Resources.

C. Cloud Federation

The Cloud Federation scenario for EE is the following: the Energy Monitor notifies about excessive energy consumption arising from a service, which is a composition of multiple cloud services. The Service Orchestrator then tries to swap some services in the service composition by searching in a Green Service Directory for iso-functional services that consume less energy than those currently being used.

A cloud federation is a multi-cloud environment that can be defined as "[*a platform that*] comprises services from different providers aggregated in a single pool" [27]. Cloud Federation tactics allow cloud-based software systems to "lease" or "negotiate" cloud services from multiple providers based on energy consumption information.

1) ***Energy Brokering****:*
Motivation. Service discoverability is one of the key principles of service orientation [28]. To enable cloud service composition in multi-cloud environments, the same principle applies. The Energy Brokering tactic makes energy information about services an additional parameter for service discovery and selection.
Description. This tactic is realized by means of two components: an Energy Broker and a Green Service Directory (GSD). The Energy Broker is a service that enables access to energy-efficient services. It receives requests for cloud services that perform a specific task and returns a pointer to the most energy-efficient service available in the multi-cloud that can perform the requested task. To do so, Energy Brokers make use of a GSD, which is a repository where all the cloud providers in the multi-cloud store the energy information of the services they provide.
Constraints. This tactic does not specify where the Energy Broker and the GSD should be hosted. However, for trust reasons, they should not be hosted by any cloud service provider participating in the federation.
Example. Gholamhosseinian et al. [29] propose a framework called Green Cloud Architecture that serves as an example for the Energy Brokering tactic. We model this example by means of a communication diagram (see Figure 3a) as a specific behavioral interaction is suggested. In the Figure, a Green Broker, instance of an Energy Broker, accepts requests for cloud services. The Broker queries the Green Offer Directory (GOD) that lists all green cloud services available. The GOD returns a list of services able to fulfill the request. The Broker then queries the Carbon Emission Directory (CED) to discover the specific EE information for each service. The combination of the CED and the GOD realize the GSD of the tactic. Finally, the Broker fulfills the request with the most energy-efficient service available.
Dependencies. The Green Service Directory has to characterize each service with its energy information. This requires Energy Indicators for each service, either static or dynamic, which suggests a dependency with either the Static Classification or the Modeling tactic, respectively.

2) ***Service-Adaptation****:*
Motivation. The main benefit of the Cloud Federation paradigm is the possibility to select services among different providers. The Energy Brokering tactic provides the energy information for services. This enables cloud-based software systems to discover services that are more energy-efficient than those currently in use. The Service-Adaptation tactic describes how Cloud platforms should switch to these more energy-efficient services.
Description. Two components realize the Service-Adaptation tactic. The first component is the Energy Orchestrator that communicates with the Energy Broker to discover energy-efficient services that fulfill a certain task and eventually performs the registration of those services with the system. This operation has to be authorized by the second component, the SLA Violation Checker, which ensures that the new services meet the service-level objectives required by the system. This component is similar to its analog in the Self-Adaptation tactics, but instead of checking the SLOs that *internal* services have to fulfill, it checks if *external* services meet the SLOs required by the system.
Constraints. The Service-Adaptation tactic assumes centralized cloud service orchestration. This creates some disadvantages in terms of flexibility because it concentrates all service orchestration logic in a single point.
Example. Villegas et al. [30] illustrate an example of the Service-Adaptation tactic in a federated cloud architecture. In their view, the Service-Adaptation is performed at the Software-as-a-Service (SaaS) layer: whenever a service request to the federated cloud cannot be fulfilled with the required service level or is too costly in terms of energy, it is forwarded to another federated cloud provider. As shown in Figure 3b, the SaaS Broker, of type Energy Orchestrator, negotiates the usage of a Green Service with other cloud providers. The Reputation of the cloud provider (of type SLA Violation Checker) determines if the provider meets the required service-level objectives. The Reputation is based on the SLA violation rate of the provider.
Dependencies. As implied by the tactic description, Service-Adaptation depends on the Energy Brokering tactic in order to retrieve the energy information of services.

V. Discussion

The Green Architectural Tactics presented in this work were explicitly formulated with reusability in mind. For this reason, we kept to a minimum the constraints that a tactic may impose on the general software architecture. When necessary, we made them explicit. For example, the Service-Adaptation tactic assumes the presence of a service orchestration mechanism; most of the Energy Monitoring tactics introduce a centralized Energy Database; Self-Adaptation tactics assume a high degree of decoupling between the virtual and the physical infrastructure. If these tactics are meant to be applied to an existing cloud-based system, software architects should consider whether these assumptions are compatible with the current architecture.

An alternate top-down design approach could be to describe our Tactics using a higher-level pattern language. An example might be the MAPE-K pattern [31]: Energy Monitoring Tactics can be adopted to implement the Monitoring and Analysis function, and Self-Adaptation can be adopted for Planning and Execution.

However, it is important to note that Green Architectural Tactics cannot generally be adopted in isolation: when introducing them in a software architecture, they might require other tactics to be adopted as well. In the previous section, we

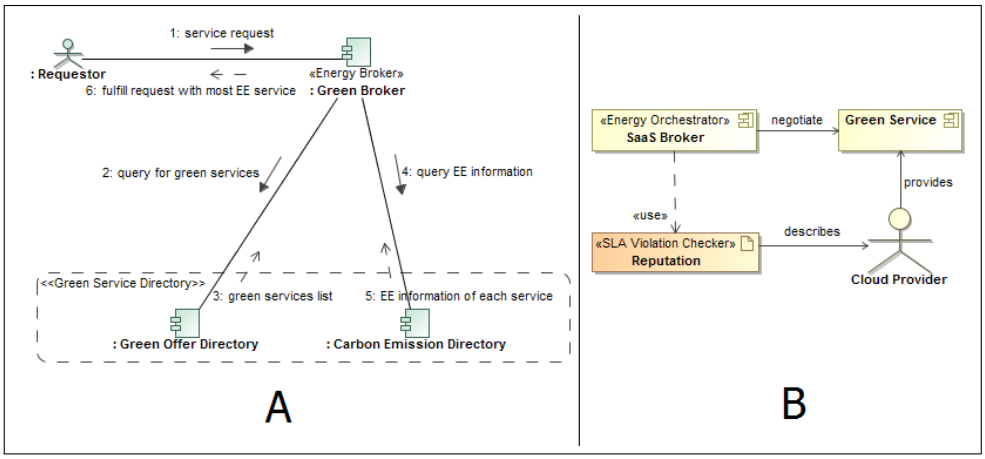

Fig. 3. A. Example of the Energy Brokering tactic. B. Example of the Service-Adaptation tactic.

TABLE II. ENERGY EFFICIENCY TRADEOFFS INTRODUCED BY GREEN ARCHITECTURAL TACTICS

Tactic	*Quality Attribute*	*Rationale*
Scaling Down	Performance	Scaling down VMs may result in lower performance in case of unanticipated demand spikes.
Consolidation	Availability Security	During VM migration some services may not be available. Live VM migration over the network requires to transfer application code, metadata and workloads, making them vulnerable to attacks.
Modeling	Modifiability	Energy Connectors are component-specific and therefore must be reimplemented if the architecture changes.
Service-Adaptation	Flexibility	The orchestrator concentrates all service composition logic in a single node.
Workload Scheduling	Performance	If workload prediction fails deadlines might be missed.

made such dependencies explicit. In short, we found that Energy Monitoring tactics are required whenever Scaling Down, Consolidation and Energy Brokering are adopted. In addition, Service Adaptation requires Energy Brokering to function properly. It is relevant to point out that the occurrence of a combination of tactics does not always imply a dependency. For example, the dependencies that emerged from our previous work [5] have identified, in a total of 26 primary studies, the following combinations: 1) Energy Monitoring and Self-Adaptation tactics, in 8 cases. 2) Self Adaptation and Cloud Federation tactics, in one case. 3) Energy Monitoring, Self-Adaptation and Cloud Federation tactics, in 2 cases. This evidence suggest a deeper relationship between the tactics that we will further explore in our future research. Furthermore, our tactics introduce tradeoffs between EE and other quality attributes, summarized in Table II. Along with the scenarios provided in Section IV, this initial trade-off analysis contributes to the identification of EE as a quality attribute. It is still under discussion to what extent EE and other sub-characteristics of environmental sustainability might influence traditional quality requirements.

VI. NEXT STEPS: TACTICS EVALUATION

Because tactics are elicited from specific implementations, they do not come with generalizable measures of the potential energy savings that they provide. In the SLR from where we extracted our tactics [6], most of the primary studies included a validation phase, performed in either an industrial or academic setting. As part of our future work, we plan to conduct research activities to provide an estimation of the impact of the adoption of the Tactics on energy consumption.

A first step will be an industrial survey among experts of the field (i.e. software architects) to have a first evaluation and prioritization of the tactics in terms of their potential impact. We already contacted a number of interested participants through our network in the Green IT Amsterdam[6] and in the EFRO MRA Cluster Green Software project[7] consortia.

Secondly, after this exploratory study, we plan to set up empirical experiments aimed at quantitatively assessing the impact of the Tactics. The experiments will be carried out on instrumented environments where we will monitor the execution of Cloud-based software applications implementing our Tactics. Meanwhile, we will gather fine-grained energy consumption data that will allow us to evaluate the energy savings gained through the Tactics implementation. For this research, we will collaborate with Cloud service providers based in Amsterdam for providing case studies and with the Hogeschool van Amsterdam (HvA) for their expertise in hardware instrumentation and measurement. We will also make use of our cluster computing resources at the VU University Amsterdam as a testbed for the experimentation.

VII. CONCLUSIONS

In cloud computing, energy efficiency aspects have to be addressed from a software architecture perspective. In this work, we provide a set of reusable design solutions, codified as *tactics*, to support the design and development of cloud-based energy efficient software. In order to help their understanding and adoption, each of our Green Architectural Tactics is presented with an example of its application extracted from the literature.

In addition, in this contribution we describe energy efficiency as a software quality attribute and analyze its archi-

[6]http://www.greenitamsterdam.nl/
[7]http://www.clustergreensoftware.nl/

tectural impact in terms of assumptions and trade-offs. In the future, we foresee the inclusion of energy efficiency in a comprehensive software quality model. For this reason, we will focus on exploring the impact of energy efficiency on other software quality attributes [32] and we will investigate guidelines and methods for assessing the energy efficiency of existing software systems.

ACKNOWLEDGMENT

This work received funding from the European Fund for Regional Development under project MRA Cluster Green Software and by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. This material has been approved for public release and unlimited distribution (DM-0001470).

REFERENCES

[1] C. Preist and P. Shabajee, "Energy Use in the Media Cloud: Behaviour Change, or Technofix?" in *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2010, pp. 581–586.

[2] E. Masanet, A. Shehabi, L. Ramakrishnan, J. Liang, X. Ma, B. Walker, V. Hendrix, and P. Maantha, "The Energy Efficiency Potential of Cloud-based Software: A U.S. Case Study," Laurence Berkeley National Lab, California, Tech. Rep., June 2013.

[3] Q. Gu, P. Lago, and S. Potenza, "Delegating data management to the cloud: a case study in a telecommunication company," in *International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA)*, vol. 7. IEEE Computer Society, sep 2013, p. 8.

[4] P. Lago, R. Kazman, N. Meyer, M. Morisio, H. A. Müller, F. Paulisch, G. Scanniello, B. Penzenstadler, and O. Zimmermann, "Exploring initial challenges for green software engineering: summary of the first GREENS workshop, at ICSE 2012," *SIGSOFT Softw. Eng. Notes*, vol. 38, no. 1, pp. 31–33, Jan. 2013.

[5] G. Procaccianti, S. Bevini, and P. Lago, "Energy efficiency in cloud software architectures," in *Proceedings of the 27th Conference on Environmental Informatics (ENVIROINFO)*, vol. 1. Shaker Verlag GmbH, 2013, pp. 291–299.

[6] G. Procaccianti, P. Lago, and S. Bevini, "Green software architectures in the cloud," *submitted to Sustainable Computing (SUSCOM), Special Issue on Software Energy Aspects of Green Computing*, 2014.

[7] G. Procaccianti, P. Lago, and G. Lewis, "Green architectural tactics for the cloud," in *Proceedings of the 11th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, Apr. 2014, to appear.

[8] E. Capra and F. Merlo, "Green IT: Everything starts from the software," in *17th European Conference on Information Systems*, S. Newell, E. A. Whitley, N. Pouloudi, J. Wareham, and L. Mathiassen, Eds., Verona, Italy, 2009, pp. 62–73.

[9] G. Rangaraj and R. Bahsoon, "Green software architectures: A market-based approach," in *The Second International Workshop on Software Research and Climate Change (WSRCC)*, 2010.

[10] R. Bahsoon, "A framework for dynamic self-optimization of power and dependability requirements in green cloud architectures," in *Proceedings of the 4th European conference on Software architecture (ECSA'10)*. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 510–514.

[11] C. Seo, G. Edwards, D. Popescu, S. Malek, and N. Medvidovic, "A framework for estimating the energy consumption induced by a distributed system's architectural style," in *Proceedings of the 8th international workshop on Specification and verification of component-based systems (SAVCBS '09)*. New York, NY, USA: ACM, 2009, pp. 27–34.

[12] S. te Brinke, S. Malakuti, C. Bockisch, L. Bergmans, and M. Akşit, "A design method for modular energy-aware software," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC '13)*. New York, NY, USA: ACM, 2013, pp. 1180–1182.

[13] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 84–106, 2013.

[14] M. Satyanarayanan, "Pervasive computing: vision and challenges," *Personal Communications, IEEE*, vol. 8, no. 4, pp. 10–17, 2001.

[15] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*, 3rd ed. Addison-Wesley, 2012.

[16] J. Carpentier, J.-P. Gelas, L. Lefevre, M. Morel, O. Mornard, and J.-P. Laisne, "CompatibleOne: Designing an Energy Efficient Open Source Cloud Broker," in *Second International Conference on Cloud and Green Computing (CGC), 2012*. IEEE, 2012, pp. 199–205.

[17] M. Alessandro, "Non-functional requirements over dynamic infrastructure services," Master's thesis, Universita' degli Studi Roma Tre, 2010.

[18] S. Reda and A. N. Nowroz, "Power modeling and characterization of computing devices," *Foundations and Trends in Electronic Design Automation*, vol. 6, no. 2, pp. 96–216, 2012.

[19] G. Procaccianti, L. Ardito, A. Vetrò, and M. Morisio, "Energy Efficiency in the ICT - Profiling Power Consumption in Desktop Computer Systems," in *Energy Efficiency - The Innovative Ways for Smart Energy, the Future Towards Modern Utilities / InTech*. Helwan: Prof. Moustafa Eissa, 2012, pp. 353–372.

[20] F. G. A. De Oliveira Jr, T. Ledoux *et al.*, "Self-optimisation of the energy footprint in service-oriented architectures," in *Proceedings of the 1st Workshop on Green Computing*, 2010, pp. 4–9.

[21] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," *Communications of the ACM*, vol. 17, no. 7, pp. 412–421, 1974.

[22] D. Rogers and U. Homann, "Application patterns for green IT," *The Architecture Journal*, vol. 18, pp. 16–21, 2008.

[23] L. Xu, G. Tan, X. Zhang, and J. Zhou, "Energy aware cloud application management in private cloud data center," in *International Conference on Cloud and Service Computing (CSC)*. IEEE, 2011, pp. 274–279.

[24] C. Dupont, G. Giuliani, F. Hermenier, T. Schulze, and A. Somov, "An energy aware framework for virtual machine placement in cloud federated data centres," in *Third International Conference on Future Energy Systems (e-Energy)*. IEEE, 2012, pp. 1–10.

[25] N. R. Herbst, S. Kounev, and R. Reussner, "Elasticity in cloud computing: What it is, and what it is not," in *10th International Conference on Autonomic Computing (ICAC 13)*. USENIX Association, June 2013, pp. 23–27.

[26] L. Lu, P. J. Varman, and K. Doshi, "Decomposing workload bursts for efficient storage resource management," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 5, pp. 860–873, 2011.

[27] T. Kurze, M. Klems, D. Bermbach, A. Lenk, S. Tai, and M. Kunze, "Cloud federation," in *The Second International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING)*, 2011, pp. 32–38.

[28] T. Erl, *SOA: principles of service design*. Prentice Hall, 2008, vol. 1.

[29] A. Gholamhosseinian and A. Khalifeh, "Cloud computing and sustainability: Energy efficiency aspects," Master's thesis, Halmstad University, 2012.

[30] D. Villegas, N. Bobroff, I. Rodero, J. Delgado, Y. Liu, A. Devarakonda, L. Fong, S. Masoud Sadjadi, and M. Parashar, "Cloud federation in a layered service model," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1330–1344, 2012.

[31] J. Kephart, J. Kephart, D. Chess, C. Boutilier, R. Das, J. O. Kephart, and W. E. Walsh, "An architectural blueprint for autonomic computing," IBM Corporation, Tech. Rep., 2005, 3rd Ed.

[32] P. Lago, S. A. Kocak, I. Crnkovic, H. Femmer, H. Muller, and B. Penzensradler, "On the quality of green software," 2013, submitted (IT Professional).